Challenges and trends for sampling-based monitoring in SDN

Catarina Pires da Silva, Solange Rito Lima and João Marco Silva

University of Minho, Department of Informatics, 4710-057 Braga, Portugal

Abstract. Network management evolved in a way where implementing complex, high level network policies, implies dealing with some attributes that depend on low-level specific configuration. This reflects on a difficulty of changing the underlying infrastructure. SDN (Software-Defined Networking) concept opens a road for new developments due to the centralized non vendor-specific control of the network, most of it related with the separation of data and control planes.

SDN raises different perspectives on how networks can operate and, consequently, how they can be managed and monitored. In particular, facing the undeniable need to reduce the amount of monitoring data in today's broadband networks, packet/flow sampling has emerged as one promising field for SDN. In this context, this paper is focused on exploring the SDN architecture, and its elements, for supporting sampling-based network monitoring. The aim is to take advantage of the integrated view of SDN controllers to apply and configure appropriate sampling techniques in network measurement points according to the requirements of specific measurement tasks. This will allow a flexible and service-oriented configuration of network monitoring, allowing also to improve the trade-off between accuracy and overhead of the monitoring process.

To pursue this, in this paper relevant SDN elements will be examined, alongside with existing monitoring solutions in the SDN research area. The analysis of these solutions led to the proposal of a new approach for the flexible configuration of sampling-based monitoring resorting to SDN components and protocols.

This flexibility also enables programmable measurements, allowing a SDN controller to manage measurement tasks concurrently at multiple spatial and temporal scales.

Since collecting actual data to create information is important at the time of taking decisions, network operators need to understand the dynamic of their network through monitoring and sampling.

Keywords: SDN, packet sampling, monitoring solutions

Acknowledgments. The heading should be treated as a subsubsection heading and should not be assigned a number.

%sectionThe References Section

1 Introduction

Despite the evolution use of traditional networks, the underneath structure of networks became less flexible. This occurs mainly due to the integration and interconnection of many proprietary, vertically integrated devices, where vendors dictate specify methods and proprietary software [1]. It can be seen from the transition from IP (Internet Protocol) version 4 to IP version 6, that is taking decades to be accomplished, or the introduction of new routing protocols, that can take a decade to become fully operational, that todays networks are rigid and somehow resilient to progress or new solutions. The use of the traditional networking architecture, means that any change will affect the entire network, so it reaches a point where networks are relatively static as their operators seek to minimize the risk of disruptions.

The SDN (Software-Defined Networking) architecture proposes to structure the network in three different layers: the infrastructure layer, the control layer and application layer. This organization has the purpose of decoupling the data and control planes allowing some networking tasks such as forwarding ruling and monitoring to be held by a centralized node called *Controller* [3]. Resorting to a centralized controller supports decision making for each device since it enables a full view of the network.

Connecting the controller to the infrastructure layer requires an API (Application Programming Interface), being the OpenFlow standard currently the most popular in the SDN domain. Then, the controller software, called NOS (Network Operating System), runs the data plane protocol so the infrastructure layer and control layer can communicate with each other, enabling networking tasks to be performed [2].

Although being an encompassing innovation enabler, currently, SDN researching is mostly focused on how to apply the architecture to fulfil todays needs related to forwarding mechanisms, scalability and security. Taking advantage of the centralized control in order to enhance monitoring and management still is an open issue, with some few works and solutions coming up in latest years.

A field with large potential to evolve through SDN is the traffic monitoring based on packet sampling. Sampling provides an overview of the network dynamics by collecting only a subset of the traffic in specific nodes, at specific time or count interval, allowing to retrieve information about what happens in the network without burden of collecting all traffic [4]. It is well documented that different sampling techniques provide distinct performance for specific task, for instance, accounting, traffic classification, intrusion detection, among others (citar minha tese). However, most of the current network devices only support a small number of techniques, namely deterministic and probabilistic sampling (detailed in Section 3).

In the context of sampling-based network monitoring, this research work explores the use of SDN concepts, elements and architecture to sustain the selection and configuration of sampling techniques in the network environment being monitored. The aim is to take advantage of SDN to provide an insight on selecting the most suitable sampling solutions for monitoring an SDN network.

Furthermore, this work introduces the elements that coexist in the network, what are the options when building a simple topology and what are the most feasible monitoring approaches, taking into account that today more and various resources are used in a network, meaning more load.

Considering the aspects above and the importance of monitoring and managing networks in a flexible and efficient way, studying a solution on how packet sampling techniques must be approached in an SDN architecture to sustain monitoring operations is the main objective to fulfill in this work.

2 Software-Defined Networking

The SDN architecture proposes a change to increase flexibility and fulfil the demands and requirements of current and next generation networks. In Figure 1, the decoupling of data and control planes is faced with the traditional architecture scheme. As illustrated, SDN provides a centralized point of control that can directly influence multiple processes of a network element using freely programmable control software. This means that we are no longer relying on proprietary management systems.

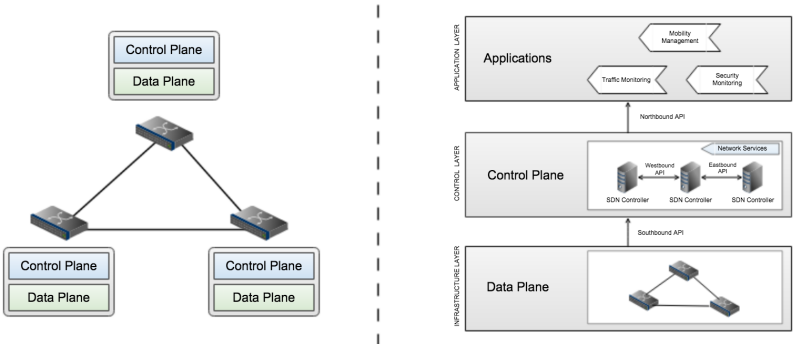

Fig. 1. Traditional Architecture vs SDN Architecture

Due to the separation of planes, the neutral software and the emerging of open and free software to control and operate networks, SDN permits the introduction of new features to be more easily implemented than in most of today's environments.

The capability of having a central point of control, accessing and viewing the whole system, while having the possibility of making different kinds of traffic engineering decisions in different regions of the network, provides an increase of flexibility on network management and monitoring [1].

2.1 SDN architecture

The SDN architecture is divided in three different layers, as illustrated in Figure 1, right side. The infrastructure layer is where networks devices such as switches and routers are, forming what is known as *data plane*. The middle layer, called *control layer*, is made by one or more *controllers*, which provide several ways of centrally operating the network. The connection between these layers is done via an API, known as *southbound* API. If it is decided to have several controllers, their connection is done via *west* and *eastbound* APIs. On the top, the application layer provides, via a northbound API, the possibility of communicating with the control layer, sending specific instructions through functional applications, which may have several purposes such as monitoring and controlling access for operation and management [3].

This separation of data and control planes is important as it becomes easier to address these two different functions and make each of them more flexible and manageable. It also allows data and control planes functions to be physically separated by hardware. Contextually, this means that an SDN switch only has a data plane module and does not have any conventional control plane functionality, fully relying on the external controller entity to make decisions. As SDN provides a more centralized control, network operators only need to manage the controllers, enabling a possible NaaS (network-as-a-service) reality.

The vision of SDN is a key enabler for simplifying management processes leading to keen interest from both the industry and the research community. Exploring the SDN architecture in network management can solve many problems because, in this way, flexibility, programmability, simplification of tasks and application deployment can be achieved through a centralized network view [6]. Managing a network with SDN means that in a single node of the network, the controller, has the power to configure, collect and store data from numerous points of the network and then analyse them.

The heterogeneous choice of SDN architecture can be observed in Figure 1. As Northbound API, the choice varies from REST(Representational State Transfer) to Procera[7] and Frenetic[8]. A variety of NOSs are available to function as SDN controllers, such as RYU [9], POX [10] and Beacon[11].

From a bottom up point of view, it first come across the Southbound API. There are several available interfaces, being OpenFlow [12] and ForCES (FORwarding & Control Element Separation) [13] the ones with greater expression.

2.2 Openflow

OpenFlow is a non-proprietary communication standard, which provides a way to establish connection between the control and infrastructure layers of an SDN architecture. Despite OpenFlow (and SDN) being used by the industry, it was initially deployed in academic campus networks [12]. It is supported by the ONF (Open Networking Foundation), which is responsible for the promotion of SDN and publication of OpenFlow switch specifications [12, 14, 15].

OpenFlow allows connection and operation of data plane, enabling a direct control of the network through setting up packet forwarding rules on network devices, such as switches. An overview of OpenFlow's scope of activity in the SDN architecture is presented in Figure 1, Infrastructure Layer.

Instructions or primitives provided by OpenFlow specifications can be used by software applications to apply rules on the SDN infrastructure layer devices. Its implementation is done on both the interfaces: at the infrastructure layer and at the control layer [12] [16].

OpenFlow specifies network traffic based on flows (i.e., packets that match the same entry in a flow table). These flows together with a set of headers, are combined with a set of admissible fields on the flow table. Flows can be defined by the control layer in a static or dynamic way.

Switches can apply rules from the OpenFlow protocol using flow tables, which are manipulated by a controller, via the OpenFlow protocol. The communication between controller and switch is done through a secure channel interface, allowing several kinds of interactions such as sending instructions from the controller to the switch or replying to a statistics request to the controller.

A flow table includes a list of flow entries used to forward packets. Each of these entries has header fields to match against incoming packets. Counters are updated for a matching packet (used for flow statistics), and actions are applied to matching packets. Actions are instructions for packet matching that allow discard, modify, queue, or forward operations to the packet.

SDN architecture working together with OpenFlow, in physical or virtual networks, gives the ability to instantly respond to changes due to the granular control provided by the OpenFlow ability of per-flow programmability [12, 17].

2.3 Open vSwitch

Open vSwitch [18] is a virtual switch consisting on a software layer that resides in a virtual machine host [19]. It was designed to bring flexibility and platform-free usage, meeting the needs of the open source community. Open vSwitch contains files with initial code from the Stanford University OpenFlow development team.

For the past several years, the focus in its development was to achieve a high level of performance in different platforms while sharing resources and workloads. To prevent problems such as consumption of hypervisor resources, Open vSwitch implements what is called flow caching [20]. Flow caching means that traffic handling is cached on the kernel module the first time a packet from a flow not handled previously, arrives at the switch. This occurs so subsequent packets that match the same flow entry do not have to be handled by the user space module again.

Today, Open vSwitch is very popular mainly due to its integration with OpenStack Networking service and it is also accepted as the genuine standard OpenFlow implementation.

Figure 2 offers an overview of the Open vSwitch architecture and its main components.

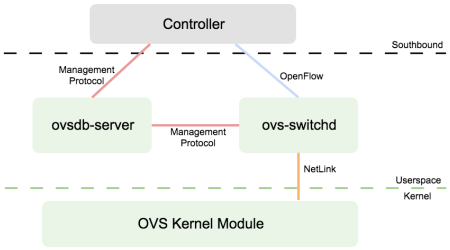

Fig. 2. Open vSwitch Architecture

The Open vSwitch kernel module uses Netlink message framing format through its AF_NETLINK sockets to access the *ovs-switchd* daemon which implements and manages all the Open vSwitch devices. The OpenFlow protocol is used to exchange messages between the Controller and *ovs-switchd*.

The datapath (*ovs kernel* module) uses Netlink socket to interact with *ovs-switchd* daemon that implements and manages any number of ovs switches on local system, and the SDN controller interacts with *ovs-switchd* using OpenFlow protocol. The *ovsdb-server* maintains the switch table database (persistent) and external clients can talk to *ovdb-sver* using JSON notation.

3 Sampling based monitoring in SDN

Network monitoring based on traffic measurement is a high demanding task due to the huge amount of data currently traversing communication infrastructures. In this way, packet sampling allows retrieving information about the whole network behaviour without the need of analysing all the data, reducing the impact of monitoring operations in the network. It is widely know that different sampling strategies lead to distinct performance in diverse network tasks. However, currently devices only offer a small set of these sampling techniques, which hampers taking full advantage of sampling features.

With the SDN architecture approach of a centralized point of control that simplifies management and manipulation tasks in the network together with OpenFlow providing ways of implementing traffic engineering. OpenFlow-based SDN are by excellence, a good way to enhance monitoring and sampling while, at the same time, providing a simplification for the introduction of new network applications. [15].

Network applications targeting monitoring and sampling can either provide new functionalities for distinct networking services or improve features previously provided by OpenFlow-based SDN. These network applications may, not only perform tasks involving network management and traffic engineering, but also tasks related to performance evaluation, network security, SLA (Service

Level Agreement) and QoS (Quality of Service) control, being the last two widely done by ISPs [21].

For better understanding traffic sampling several concepts should be considered, such as the interval between samples and sample size, described bellow.

- Sample selected network packets used for network parameters estimation. Can also be referred as an individual action of selecting and capturing packets from the stream;
- Sample size number of packets selected and captured to constitute a sample. It can also be a time interval. Sample size is controlled by triggers that delimit size by packet position into the stream or timestamp;
- Interval between samples Number or time interval of ignored packets of a stream. Analogous to the sample size, it is also controlled by triggers.

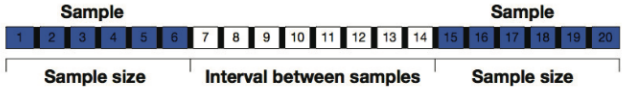

Fig. 3. Sampling Concepts [23]

The main sampling techniques, deployed or not in network devices, resort to selection processes using the packet position, through counters or timestamp, from the stream under observation. These techniques are briefly detailed bellow:

Systematic Count-based

The starting point of a sample and sampling size are operated by the spatial packet position (resorting to packet counters) using a deterministic function that results in a periodic behaviour.

Systematic Time-based

The systematic time-based sampling technique, similarly to the systematic time-based, also used a deterministic function to rule the sample size and interval between samples. The difference resides in the type of triggers: in this technique, they are oriented by the packet arrival time [23].

Random n-out-of-N

The packet selection is ruled by a random process, being the simplest and widely deployed mechanism the capture of n packets from a sequential stream of N packets (n-out-of-N). A pseudorandom function generates n numbers (between [1,N]). Then the packets that have a position equal to one of the random numbers are selected and captured [23]. The probability p (with $p = n/N$) is applied for all the N packets to be selected and compose the sample.

Uniform probabilistic

A predefined uniform probabilistic function decides the packet selection to compose a sample, having all packets the same probability of being selected.

An example of a random uniform probabilistic technique is a count-driven technique with an independent random variable with distribution of mean 1/p and successive intervals between samples (with sample size equal to 1 packet) [23].

3.1 Sampling tools for SDNs

Although promising, the use of traffic sampling along with SDN concepts and architecture is exploited for only few tools, namely sFlow and FleXam.

sFlow ONF (Open Networking Foundation) is focused not only on the dissemination and development of the OpenFlow standard on the network industry, as many members of the ONF are major network operators and manufacturers. Some of these members are shared with the sFlow.org industry consortium that has similar objectives for the sFlow standard, making its support available in OpenFlow and non-Openflow switches [24].

sFlow proposes that operations such as monitoring no longer be implemented on the switch, instead sampled packet headers are sent to a separate component of the control plane, called monitor, that gets packet headers, decodes them and aggregates the data through a traffic analysis application [25].

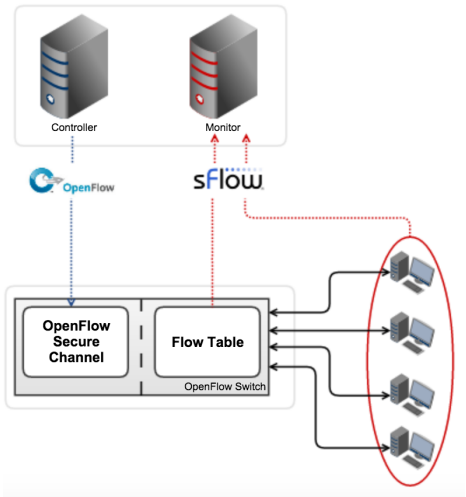

Fig. 4. sFlow OpenFlow-based SDN Architecture

As it can be noticed on Figure 4, sFlow and OpenFlow work in a partnership. It is intended that the controller, using OpenFlow, configures the forwarding tables in switches and sFlow increases the visibility by providing real-time access into traffic that flows in the network. Having this type of visibility means that

the network can adapt to changing demands [26] [24]. This represents the use of sFlow when packet forwarding is controlled by OpenFlow.

One major problem regarding sFlow is that its reports do not include the entire packet, which can be a problem when more packet information is required. In addition, it only provides uniform sampling methods [21].

FleXam is a per-flow sampling extension for the OpenFlow standard, allowing the controller to access packet-level information. Its priority is to overcome some problems, which may arise in the use of the OpenFlow alone. One of these problems is the increase of flow-entries. From OpenFlow version 1.0 to, for instance, OpenFlow version 1.2, a flow entry went from a 12-tuple match to OXM (OpenFlow Extensible Match) based on TLV (Type-length-value) structures, that allows switches to support a wider range of header fields (for instance, OpenFlow 1.4 supports 41 different types, where TLV was also added to ports, tables and queues) [21, 27, 28].

Packets in FleXam can be sampled stochastically, meaning that a predetermined probability is set, or deterministically, which implies a pattern. This flexibility in sampling is enhanced by the possibility of the controller to define several rules on the packets, such as which should be sampled, what part of it should be selected and where they should be sent [27].

FleXam was implemented as a patch to Open vSwitch and enables the access to packet-level information at the controller where an application should run, allowing the installation of rules, processing sampled packets and collect information.

This sampling extension, in addition to presenting itself with two sampling techniques, is considered flexible for some reasons such as providing a stochastic sampling and a generalized version of the deterministic sampling. The stochastic sampling consists in the selection of packets that are included in a flow, with a probability of p. On the other hand, the generalized version of the deterministic sampling is formulated as selecting m consecutive packets from each k consecutive packets, ignoring the first δ packets.

The downside of FleXam when compared to sFlow and other sampling tools is that it offers only a per-flow sampling, without the ability of performing per-packet sampling.

4 Enhancing Sampling in SDN

In this section all the previously presented elements are taken in consideration and the most suitable solution for implementing sampling-based monitoring in SDN is discussed. As a first approach, using what already exists and make it work such as it is intended is, most of the times, considered and it offers a viable solution. Moreover, a deeper evaluation of how the network elements interact is presented and conclusions are drawn.

4.1 Design Goals

To propose a solution that better explores and fits the SDN architecture some items were defined as design goals:

- Compatibility with popular software for SDN;
- Efficient and lightweight implementation without compromising the SDN proposal;
- Explore existing solutions of data and control planes and attach monitoring to them without the need of brand new software;
- Open and standard protocols/software sustention.

4.2 Interaction Between Elements

The goal is to perform packet sampling through count or time intervals using the Open-Flow specification provides. The concern of knowing or having control of what happens in the network begun with the first QoS implementation in OpenFlow 0.8.0, but it was not until version 1.3 that OpenFlow substantially increased its QoS framework functionality.

After implementing this kind of mechanism there is still work to do in order to sampling and analysing traffic, particularly at packet-level. In this scope, tools such as FleXam are used, as OpenFlow itself has nothing to provide. In this work, it is intended to discuss packet-level sampling applying two possible approaches: one from the controller to switch and the other from the switch to the controller. Each strategy is characterized by where and who controls the rules applied for packet selection.

Controller to Switch Here, the controller is responsible for making sure the sampling intervals are accomplished and the packet information is stored where it should be. This means that the controller is responsible for managing the rules of sampling, with the switch not being aware that a specific selection is being made, because it is the controller who, in some way, forces that. An example consists in a controller requesting a specific subset of packets. A generic representation is shown in Figure 5.

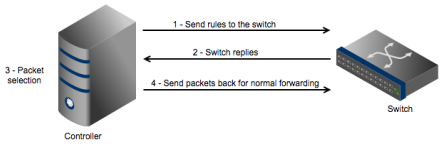

Fig. 5. Controller to Switch - Minimal Approach

In this context, there were two solutions that appeared to be the best path to be taken:

1. Sending all packets, being them original packets, or copies of the packets, matching a flow entry, to the controller [12]. In this solution, every packet is forwarded to the controller, where they are counted. When the counter reaches the intended value, the packet is collected. If the packet is not supposed to be collected, there are two options, depending if the packet sent by the switch is the original or a copy. If it is the original packet, the controller must send a *Packet-Out* message containing the packet, injecting the packet in the data plane. In the other hand, if it is a copy of the packet that is being handled, discarding the packet is the normal procedure.
   Problems identified in this solution:
   - Accumulated traffic in the controller;
   - Possible bottleneck in the controller;
   - Possible delays in the packet forwarding to the switch (this is not a problem if only a copy of the packet is delivered);
   - Possible packet-loss between the controller and switch communications;
2. Request flow statistics from the switch within a pre-set time interval. In case the statistics packet counter is next to the intended value, change the rule to send the packet to the controller. This solution can fit both time-based and count-based sampling.
   In the OpenFlow specifications, statistics can be demanded from a controller. There are several types of statistics that can be requested such as flow and table statistics. This exchange of information starts by a request message from the controller, mentioning what statistics are wanted, and is followed by a reply from the switch containing the requested information, implying two interactions to get the statistics.
   The idea is having a thread which is launched from the controller, from time to time, making a flow statistics request to the switch. The flow statistics reply from the switch includes information such as the number of packets in flow. This value is the one which the controller should pay attention to as it comes closer to the value pretended. The procedure behaves as follows: a *Flow Modify* message would be sent to the switch, followed by a *Packet-In* message sent to the controller, so that packets, matching that flow, are forwarded to the controller. When the packet arrives, it is collected. After that, another *Flow Modify* message is sent to the switch and, as the packets are forwarded normally, counters are reset.
   Problems identified in this solution:
   - Statistics requests may not match with counter values intended since the communication between requesting statistics and responding to them implies delay.
   - To obtain higher accuracy with this solution, a previous knowledge of the network dynamic is essential and, even though it can be obtainable, it does not guarantee good levels of synchronism.

There could be similar solutions to the ones presented here but all of them rely in the fact that, to have packet selection/collection, several messages between the controller and switch communication are involved, which ultimately leads to lack of performance.

Switch to Controller Conversely, switch to controller interaction consists in applying rules directly on the switch. In this situation, the role of the controller is only to set the parameters of sampling required by the application, sending it to the switch, which in turn, after receive it, will apply it accordingly.

Since it is the switch the one to apply the parameters, this means that switch will be responsible by the selection and collection of packets, and responsible to send them automatically to where the controller ordered. It can be to a monitor or to the controller itself.

To summarize, the controller will only fill a rule with parameters to the switch and the switch will do the whole work and then redirect the outcome. In Figure 6, a graphic scheme of the interaction is represented.

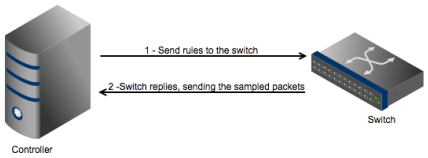

Fig. 6. Switch to Controller - Minimal Approach

After considering the OpenFlow standard, it comes clear that the option that could produce a better solution is to implement a sampling mechanism within the OpenFlow itself by biding the operations on the switch, as presented in this subsection.

4.3 Proposed Method

The wiser approach for a new solution to control sampling processes in SDN environments is to create custom actions in the OpenFlow switch to implement the sampling rules. Those customized actions would be added in the form of patches to the OpenFlow specification, resembling FleXams implementation. Opposing to FleXam's approach, which unifies some sampling techniques in a single action, the ideal solution for this work would be that a single action corresponds to a single sampling technique.

For this patch, the OpenFlow version to be used must be at least version 1.0. The reason behind this choice is that OpenFlow version 1.0 was considered as the unified version from which all vendors should start adopting the OpenFlow standard, resulting in a large software support for version 1.0. If the patch is not implemented in the OpenFlow version 1.0, but instead on a newer version, the concern is what software should be used to provide support to work with that version.

The first sampling technique to be implemented (represented in Figure 9) will be a count-based sampling, where packets will be counted and then sent to the controller for storage, following selected parameters.

Fig. 7. Switch to Controller - Minimal Approach

OFPAT_INTERVALC represents the action name, that includes two parameters: interface and interval. The interface parameter indicates on which switch interface will the incoming packets be sampled and interval represents the counting interval in which the packets will be selected as sample.

Open vSwitch will be the OpenFlow implementation software where the path will be developed. Since it is largely used by the OpenFlow community, is widely supported from SDN software, has a solid OpenFlow implementation and there is documentation available about how it works. Open vSwitch offers the possibility to have a usable and practical developed patch.

For this implementation, it is intended to add a counter field in the packet code structure, first in the userspace datapath of Open vSwitch, and then transport it to the kernel module. With the counting implemented, it is time to implement a rule for all packets not selected for sampling to be forwarded to the right path. Selected packets should be duplicated with one copy being sent to the controller for sampling and another copy to be normally processed and forwarded.

The environment in which this work will be done is virtual. The Mininet[29] network emulator allows a user to run several networking elements, such as virtual hosts and switches, using the same Linux kernel. Mininets particularity of having switches supporting Openflow and SDN systems, alongside its popularity, was what raised interest in using it for this work.

An OpenFlows switch implementation will be used. A brief approach to what is available is required as, to make possible the production of results, the most suitable for this work is too the most used OpenFlow's switch version at the moment, which is Open Flow version 1.3.

5 Ongoing Work

The ongoing work for the proposed solution consists on the development, implementation and testing of sampling methods. A model of how the first sampling method can be built is presented on Chapter 4. To make a first functional patch the following steps are recommended:

1. Implement the sampling action on the Open vSwitch userspace datapath;
2. Test its functionality through the DPCTL tool, which monitors and administrates OpenFlow datapaths;
3. To have it working on a real network environment, add a patch to a NOS.

6 Conclusion

One of the most important aspects of the SDN architecture is to favour the introduction of new concepts and its high programmability. This ability to facilitate the changes in operations is mainly due to the separation between the control and data layer, allowing a single control on several data elements in the network.

Monitoring and sampling are essential tasks to perform in any system, and a system that works under an SDN concept is no different. While taking advantage of SDN features, the goal is to introduce sampling techniques that will allow to monitor the network. The data layer SDN protocol OpenFlow focus solely on the task of forwarding packets meaning that no concerns about monitoring the state of the network were taken into account when this protocol was designed. However, there are some tools based on this protocol to perform this kind of tasks, being the most known sFlow.

To have SDN working on a network, a control module is mandatory, so the rules can be sent to the data plane to be applied.

This proposal of development emerged after realizing that none of the existing tools executes monitoring tasks as intended. To begin, an approach based on not changing any of the network elements standard operations was made. However, this solution was considered invalid given that the performance decrease would not allow the solution to work on a real network environment.

Finding a solution on what to do to avoid this problem required a deconstruction of how the elements operate, and the conclusion was that, for benefit of the solution, sampling operations had to use the data layer.

As previously said, the solution to be developed must consider that OpenFlow has limited resources when it comes to monitoring tasks, including sampling.

Throughout the use of SDN network emulator Mininet and, the OpenFlow implementation, Open vSwitch, both ensuring high programmability of features, this work proposes the implementation of a patch to the OpenFlow protocol that enables the sampling of packets, beginning by doing it so in a per-packet count-based sampling method.

References

1. Kim, H., Feamster, N.: Improving network management with software defined networking. IEEE Communications Magazine **51** (2013) 114–119
2. Open Networking Foundation: Software-Defined Networking: The New Norm for Networks. ONF White Paper (2012) 1–12

3. Sezer, S., Scott-Hayward, S., Chouhan, P., Fraser, B., Lake, D., Finnegan, J., Viljoen, N., Miller, M., Rao, N.: Are we ready for SDN? Implementation challenges for software-defined networks. IEEE Communications Magazine **51** (2013) 36–43
4. Duffield, N.: Sampling for passive internet measurement: A review. Statistical Science **19** (2004) 472–498
5. T. Zseby, T.H., Claise, B.: Packet sampling for flow accounting: Challenges and limitations. Lecture Notes in Computer Science **vol. 4979** (2008) 6171
6. Tuncer, D., Charalambides, M., Clayman, S., Pavlou, G.: Adaptive Resource Management and Control in Software Defined Networks. IEEE Transactions on Network and Service Management **12** (2015) 18–33
7. Voellmy, A., Kim, H., Feamster, N.: Procera: A language for high-level reactive network control. HotSDN '12, New York, NY, USA, ACM (2012) 43–48
8. Foster, N., Harrison, R., Freedman, M.J., Monsanto, C., Rexford, J., Story, A., Walker, D.: Frenetic: A network programming language. ICFP '11, New York, NY, USA, ACM (2011) 279–291
9. Telegraph, N., Corporation, T.: Ryu network operating system. https://osrg.github.io/ryu/ (2016)
10. McCauley, M.: Pox. https://github.com/noxrepo/pox (2016)
11. Erickson, D.: The beacon openflow controller. Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (2013) 13–18
12. McKeown, N.: OpenFlow: Enabling Innovation in Campus Networks (2008)
13. Doria, A., Salim, J.H., Haas, R., Wang, W., Dong, L., Gopal, R.: Forwarding and control element separation (forces) protocol specification. Technical report (2010)
14. Halpern, J., Salim, J.H.: Software-Defined Networking : Experimenting with the control to forwarding plane interface Extending the OpenFlow protocol with ForCES concepts . 2012 European Workshop on Software Defined Networking (2012) 91–96
15. Kreutz, D., Ramos, F.M.V., Verissimo, P., Rothenberg, C.E., Azodolmolky, S., Uhlig, S., Kreutz, D., Ramos, F.: (Software-Defined Networking: A Comprehensive Survey) 1–61
16. Heller, B.: OpenFlow Switch Specification v1.0.0. Current **0** (2009) 1–36
17. Blaiech, K., Hamadi, S., Valtchev, P., Cherkaoui, O., Beliveau, A.: Toward a semantic-based packet forwarding model for openflow. In: Network Softwarization (NetSoft), 2015 1st IEEE Conference on, IEEE (2015) 1–6
18. Openvswitch: Open vswitch. (https://github.com/openvswitch/ovs)
19. : Open vswitch. (http://openvswitch.org/)
20. Pfaff, B., Pettit, J., Koponen, T., Jackson, E.J., Zhou, A., Rajahalme, J., Gross, J., Wang, A., Stringer, J., Shelar, P., Amidon, K., Casado, M.: The design and implementation of open vswitch. Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (2015) 117–130
21. Shirali-Shahreza, S., Ganjali, Y.: Efficient Implementation of Security Applications in OpenFlow Controller with FleXam. In: 2013 IEEE 21st Annual Symposium on High-Performance Interconnects, IEEE (2013) 49–54
22. Joo Marco C. Silva, Paulo Carvalho, S.R.L.: Inside packet sampling techniques: exploring modularity to enhance network measurements. (29 March 2016)
23. Silva, J.M.C.: A modular traffic sampling architecture for flexible network measurements. Doctoral thesis, Universidade do Minho, Braga, Portugal (2015)
24. Giotis, K., Argyropoulos, C., Androulidakis, G., Kalogeras, D., Maglaris, V.: Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments. Computer Networks **62** (2014) 122–136

25. Phaal, P.: Software defined networking. http://blog.sflow.com/2012/05/software-defined-networking.html (2012)
26. Phaal, P.: Openflow and sflow. http://blog.sflow.com/2011/05/openflow-and-sflow.html (2011)
27. Shirali-Shahreza, S., Ganjali, Y.: FleXam: Flexible Sampling Extension for Monitoring and Security Applications in OpenFlow. Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking - HotSDN '13 (2013) 167
28. Shirali-Shahreza, S., Ganjali, Y.: Traffic statistics collection with FleXam. Proceedings of the 2014 ACM conference on SIGCOMM - SIGCOMM '14 (2014) 117–118
29. Lantz, B., Heller, B., McKeown, N.: A network in a laptop: Rapid prototyping for software-defined networks. (2010) 19:1–19:6